Narrowing the Synthesis Gap: Academic FPGA Synthesis is Catching Up With the Industry

Benjamin L.C. Barzen*, Arya Reais-Parsi*, Eddie Hung†, Minwoo Kang*,
Alan Mishchenko*, Jonathan W. Greene* and John Wawrzynek*
* *Department of EECS, University of California*; Berkeley, USA
{bbarzen,aryap,minwoo_kang,alanmi,johnw}@berkeley.edu, jgreene@cambioscomputing.com
† *FPG-eh Research* and *University of British Columbia*; Vancouver, Canada
eddie@fpgeh.com

***Abstract*—Historically, open-source FPGA synthesis and technology mapping tools have been considered far inferior to industry-standard tools. We show that this is no longer true. Improvements in recent years to Yosys (Verilog elaborator) and ABC (technology mapper) have resulted in substantially better performance, evident in both the reduction of area utilization and the increase in the maximum achievable clock frequency. More specifically, we describe how ABC9 — a set of feature additions to ABC — was integrated into Yosys upstream and available in the latest version. Technology mapping now has a complete view of the circuit, including support for hard blocks (e.g., carry chains) and multiple clock domains for timing-aware mapping. We demonstrate how these improvements accumulate in dramatically better synthesis results, with Yosys-ABC9 reducing the delay gap from 30% to 0% on a commercial FPGA target for the commonly used VTR benchmark, thus matching Vivado's performance in terms of maximum clock frequency. We also measured the performance on a selection of circuits from OpenCores as well as literature, comparing the results produced by Vivado, Yosys-ABC1 (existing work), and the proposed Yosys-ABC9 integration.**

***Index Terms*—Field programmable gate arrays, synthesis, technology mapping, Hardware design languages, Table lookup**

I. Introduction

In the FPGA developer community, open-source tools are generally considered inferior to their closed-source counterparts. Most notably, the 2015 "Mind The Gap" [1] paper showed that open-source/academic flows (Yosys [2], ABC [3], Verilog-To-Routing/VPR [4]) performed consistently worse than a closed-source industry-standard flow (then Xilinx ISE) in terms of critical-path delay, area utilization and compilation run-time. A key conclusion of that work was that the gap in critical-path delay lies primarily in the front-end synthesis and technology mapping stages (explained in Section II), comprising 31 percentage points (pp) of the measured deficit. The delay gap attributable to the back-end stage steps of packing, placement and routing, meanwhile, contributed to 25 pp altogether. The work further noted that proprietary flows spend more than three times as long as open-source alternatives in the front-end stages (at 13% and 4% of their overall run times, respectively), suggesting that academic flows could afford to expend a lot more effort there.

In this paper, we revisit the comparison of open-source and proprietary tools. Specifically, we note that considerable work has been done by the open-source community since [1] to address this so-called *synthesis gap*. An integral part of this effort has been the development of "ABC9", a set of feature additions to ABC [3] to improve logic synthesis and technology mapping. Xilinx's ISE has now also been superseded by Vivado, a modernized tool flow for Xilinx (now part of AMD) devices. Hence motivated, we re-evaluate the academic/proprietary divide to better inform the direction of future work.

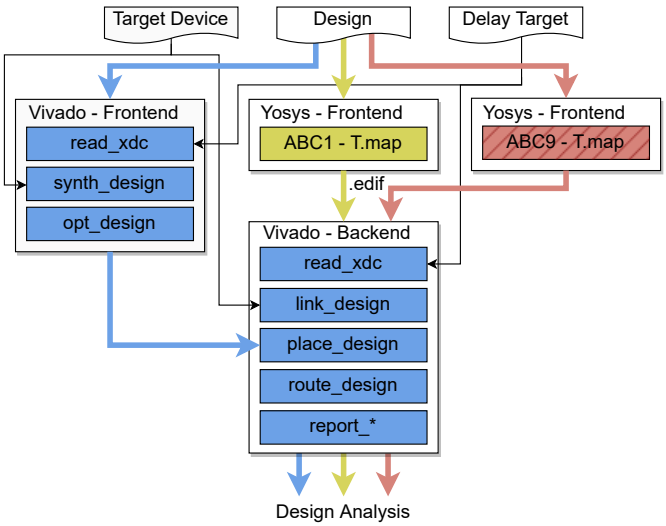
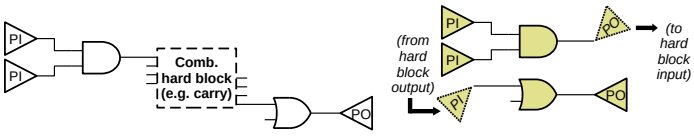


Fig. 1: Flows examined in this paper. Blue refers to Vivado, yellow to Yosys-ABC1 (existing work), and red to Yosys-ABC9 (proposed).

In the following sections we discuss the salient features of ABC9 by comparing it against the existing baseline ABC1, and show how these features help to improve synthesis quality. We conduct a series of experiments to showcase the improvements of the proposed open-source synthesis flow, Yosys-ABC9, against prior work, Yosys-ABC1. We compare both against the industry standard baseline flow Vivado. (Figure 1 summarizes these flows.) Finally, we enumerate promising next-steps for furthering the academic state-of-the-art and conclude.

II. ABC1: Cornerstone of Existing Flows

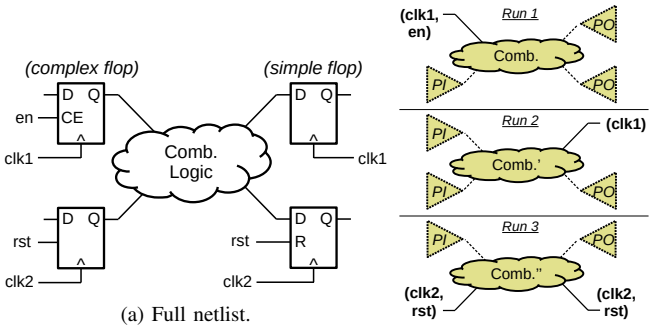
An FPGA compilation flow is typically divided into front-end and back-end stages, as shown in Fig. 1. The front-end, on which this work focuses, comprises two important



(a) True path.

(b) Truncated path as viewed by ABC1.

Fig. 2: Treatment of a path through a simple circuit, starting at an I/O pad (primary input), passing through an AND gate, then a hard-block, and then into an OR gate before reaching another I/O pad (primary output).



(a) Full netlist.

(b) Truncated netlist supplied to ABC1 over multiple runs.

Fig. 3: Example netlist containing simple and complex flops.

steps: *synthesis* and *technology mapping*. Synthesis is the process of transforming an RTL specification (e.g., in Verilog) into a circuit netlist consisting of a) technology-*independent* logic primitives (e.g., 2-input NAND gates); b) technology-*dependent* state elements (e.g., flip-flops with clock-enable); and, c) hardened IP blocks appropriate for a target device (e.g., the carry chains, DSPs and block RAMs in modern FPGAs.) Within popular academic flows this step is performed by ABC (in Yosys [2] by its `abc` pass) or ODIN-II (in Verilog-to-Routing, VTR [4]).

Technology mapping describes the specialization of this netlist into a fully technology-*dependent* circuit, replacing all generic logic primitives with an implementation in FPGA lookup tables (LUTs). Typically, the objective of a technology mapper is to find a solution that minimizes critical-path delay while using the smallest area possible. ABC is used for this step in both Yosys- and VTR-based academic flows. (ABC has also reportedly seen adoption by commercial FPGA vendors).

Lastly, the back-end stage steps of placement and routing take this technology-dependent netlist and compute a non-overlapping physical arrangement of all components, including connecting wires, for realization on a target FPGA.

ABC provides a toolbox of technology-independent optimization passes (including structural hashing, functional reduction, SAT sweeping [5], etc.) and technology-dependent LUT-mapping techniques that can be variously assembled into flows. Previously, one such flow could be found in both Yosys and VTR; we refer to it as "ABC1"[1].

Limitations of ABC1

The incumbent suffers two key limitations:

1) Hard Blocks: ABC1 does not support combinatorial "hard blocks" (fixed-function, opaque circuit modules such as adder-chains, multipliers, memories, etc.). Instead, it works with a monolithic logic cone, eliding any hard-blocks and replacing their connections with external ("primary") inputs and outputs. (This is just as it would be if they were driven by an opaque, off-chip source, per Figure 2.) Subsequently, ABC1's optimization passes are unable to integrate analysis of these blocks, such as to propagate constants and don't-cares through them. This might otherwise enable outputs to be replaced by constants from the input or, in the case where *all* block outputs can be replaced with constants from (or constant functions of) the inputs, enable the entire hard-block to be replaced with a simple equivalent circuit.

Moreover, delay propagation through hard blocks is not taken into account. Consider for example the path from Fig. 2 as being the critical path of the circuit, and assume that the delay of each primitive is 1 unit. In this case ABC1's view of the circuit would be composed of two truncated paths of 1 unit delay each, instead of the true critical-path delay of 3 units; accordingly it may over-optimize the design by expending more area for no actual gain in delay.

Worse still, since functional information about the replacement primary inputs is not known, logic synthesis cannot prove any equivalences between the block's corresponding outputs and other nodes. At best, this process of equivalent node detection and merging (known as SAT-sweeping [5]) would otherwise be able to find and remove whole redundant blocks, as it often does in some designs.

2) Complex Flops: A second limitation of the ABC1 flow is that it can support neither *sequential synthesis* on designs with flip-flops from multiple clock domains, nor the complex flip-flops present on modern devices with clock-enable and synchronous/asynchronous set-reset capabilities. *Sequential synthesis* describes a class of optimizations that examine both the combinatorial and sequential behavior of a circuit to further eliminate redundancy: for example, were the mapper able to prove that two flip-flops are always inversions of each other, one flop could be discarded and its output driven from an inversion of the remaining flop.

Since ABC1 cannot understand flip-flops belonging to more than a single clock domain (improbable for advanced designs), this necessitates multiple ABC1 runs in sequence, each containing the latest netlist (or the relevant subset thereof) with all incompatible flops abstracted out of the circuit. Yosys-ABC1 takes this workaround a step further for complex flip-flops, using a separate run for each "control set" (the tuple of clock domain plus enable/set/reset input signals) as shown in Fig. 3.

III. ABC9: Addressing ABC1's Shortcomings

ABC9 builds on ABC1 by addressing the limitations described above. It adds support for the combinatorial hard-blocks (prevalent in modern FPGAs; this support makes a

[1]We write "ABC" to refer to the multi-purpose synthesis and mapping toolbox, and "ABC9" to refer to the improved flow described in Section III.

range of other optimizations more effective, including retiming.), complex flip-flops across multiple clock domains and control signals, and full timing awareness. Additionally, a number of runtime and memory improvements in ABC9 allow larger circuits to be processed. In this section, we describe how ABC9 was integrated into open-source Yosys as its `abc9` pass using a flexible data-driven approach to easily enable support for current and future FPGA architectures.

A. White-Box Support for Hard Blocks

ABC9 allows the logical contents of a hard-block to be communicated alongside the user netlist, transforming an opaque black-box primitive into a transparent box through which the missing optimizations (described in Section II-1) can be applied. The complete logic description of the box is available on-demand for simulation and computing local don't-cares during post-mapping re-synthesis. The resulting boxes are called "white-boxes" since their functionality and timing information are available. The majority of optimization engines in ABC9 have access to this information.

Yosys supplies its own reference Verilog models for all supported primitives, primarily for simulation and verification purposes. During synthesis, typically only the primitive interface (i.e., the set of input and output ports) is used to ensure correct usage. Since the behavior of a hard-block cannot be modified, and since Yosys cannot perform cross-module optimizations without flattening the design (lest the logic be realized in soft LUTs), such primitives can be treated as black-boxes. In Yosys-ABC9, these same reference models are leveraged to allow users to easily indicate which primitives should be elaborated. They are then communicated as white-boxes to ABC9 using Verilog attributes, as shown in Listing 1:

```
(* abc9_box, lib_whitebox *) // Instances of this module are
                             // to be an ABC9 white-box
module CARRY4(
  (* abc9_carry *) output [3:0] CO,
  // ^ Attribute above marks a special carry-in or out port
  //   to ensure that a chain of carry blocks are not broken
                   output [3:0] O,
  (* abc9_carry *) input        CI,
                   input        CYINIT,
                   input  [3:0] DI, S
);
  // Following behavior to be elaborated and provided to ABC9
  assign O = S ^ {CO[2:0], CI | CYINIT};
  assign CO[0] = S[0] ? CI | CYINIT : DI[0];
  assign CO[1] = S[1] ? CO[0] : DI[1];
  assign CO[2] = S[2] ? CO[1] : DI[2];
  assign CO[3] = S[3] ? CO[2] : DI[3];
  ...
endmodule
```

Listing 1: Example Yosys simulation model for carry hard-block, annotated with Verilog attributes to indicate ABC9 white-box eligibility. Not shown: modules with behavior dependent on Verilog parameters are also supported.

B. Complex Flop Support

ABC9 adds support for complex flops by allowing combinatorial white-boxes (the contents of which are also inferred from a Verilog model) to be attached to the input of a simple flop as shown in Figure 4. This allows any synchronous functionality—including enables, sets and resets—to be described. Additionally, the clock domain of each flop in the netlist can also be provided to ABC9, allowing sequential optimization passes to operate on the entire circuit (negating the need for multiple redundant ABC1 invocations) as well as to merge only flops sensitive to the same clock signal and edge.

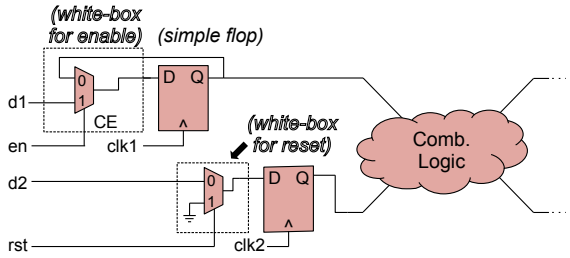


Fig. 4: ABC9's support for complex flops through extracting clock enable/set/reset functionality into white-boxes attached to inputs of simple flops.

C. Timing-Aware Mapping

With ABC9 now able to understand the entire design netlist, without any truncation or partitioning, there exists the opportunity to perform timing-aware mapping effectively. Although ABC1 does support the specification of real delay numbers for LUT inputs and ABC1 will optimize for the minimum path delay, its incomplete view of the design meant that this process was of limited effectiveness over the simpler metric of optimizing for minimum LUT depth.

ABC9 continues to support real delay numbers for LUT inputs, and extends this to white-boxes. Similar to leveraging the Verilog simulation model for describing the behavior of hard-blocks, LUT and white-box delays now can be provided to these same Verilog models as shown below:

```
(* abc9_box, lib_whitebox *)
module CARRY4(
  (* ... *) output [3:0] CO,   output [3:0] O,
  (* ... *) input        CI,   input        CYINIT,
             input  [3:0] DI, S
);
  ...
  specify // Non-synthesizable Verilog-2001 construct
          // typically used during simulation
    (CYINIT => O[0]) = 111; // Simple path
    (S[0]   => O[0]) = 222;
    (CI[0]  => O[0]) = 333;
    ...
  endspecify
endmodule
```

Listing 2: Example Yosys simulation model for carry hard-block, extended with timing annotations. Units are implicit, Yosys' models uses picoseconds.

Beyond support for combinatorial delays, ABC9 also supports the imposition of setup and clock-to-Q delays (referring to them respectively as required and arrival times), which can be significant for larger hard-blocks such as block RAMs. Similarly, such delays can also be provided within the `specify` block using `$setup()` and edge-sensitive path constructs. Yosys-ABC9 is able to interpret this information and communicate it to ABC9 in the format it expects so that it is able to minimize the true critical-path delay, rather than optimizing for the minimum LUT depth of a possibly truncated path as done previously.

D. Area Recovery

Since ABC9 has a full picture of the functional behavior of all soft-logic and hard-blocks in the design along with an

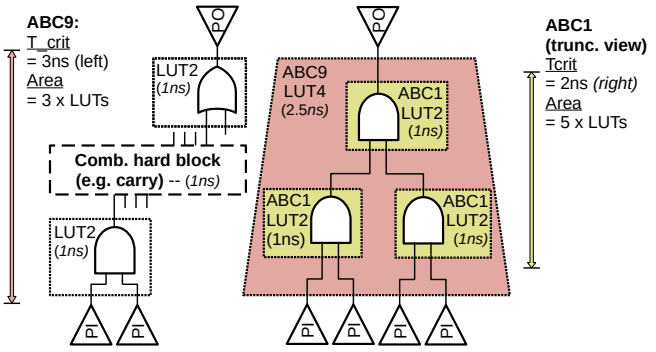


Fig. 5: Illustration of ABC9's area recovery effectiveness, since its non-truncated view of the design netlist allows identification of the true critical-path, thus choosing to use fewer (slower) LUTs to implement the design.

understanding of the timing characteristics of all internal paths, mapping optimizations can now be applied more effectively. One noteworthy optimization is area recovery, which is the ability to use slower but more area-efficient resources on non critical-paths. An example of this is shown in Figure 5 where fewer, larger, and slower LUTs are used to implement paths as long as they do not increase the critical-path delay.

E. Efficiency Improvements

A number of efficiency improvements have been made to ABC9 to make it more scalable than ABC1, despite the additional complexity of white-box and timing support. Yosys-ABC9 is able to transform its internal representation (RTLIL) of the design netlist and all white-boxes into And-Inverter Graph (AIG) form, along with extensions describing other complex-flop information, and pass that into ABC9 as well as to parse and stitch its output back into Yosys' RTLIL.

F. Structural Choices

The use of structural choices was introduced 25 years ago [6]. Currently it is one of the main features accounting for improved quality of results produced by ABC, compared to other logic synthesis tools. Structural choices are derived by considering several functionally equivalent but structurally different versions of the logic cloud. Fast equivalence checking based on SAT sweeping [5] can be used to detect equivalent nodes. Rather than merging equivalences and keeping only one circuit structure, the use of structural choices assumed that all circuit structures are kept and passed to the technology mapper [7].

The availability of structural choices allows the mapper to use delay-optimized circuit structures along the critical path and area-optimized structures elsewhere, improving both area and delay. The runtime for computing structural choices and using them in the mapper tends to be modest when resource limits are enforced. Thus, the SAT solver times out on hard-to-prove equivalences, which leads to fewer choices but results in reasonable runtime.

IV. Methodology

Experiments were conducted using 3 different flows: Vivado, Yosys-ABC1 and Yosys-ABC9, shown in Figure 1. The Vivado flow consists of one program, Vivado v2022.1, which performs synthesis, technology mapping as well as place and route. For the Yosys(v0.17)-ABC1 and -ABC9 flows, synthesis and technology mapping is performed by Yosys and ABC respectively, while place and route is always performed by Vivado. The Vivado and Yosys flows are used with default settings, i.e., no parameter optimization is conducted. The exact Vivado and Yosys commands can be obtained from our GitHub repository [8]. Since Yosys and ABC can only optimize for the best possible delay rather than a specific period constraint (a topic for future work) frontend synthesis needs only to be run once each for Yosys-ABC1 and -ABC9. For the remaining part of the Yosys flows, our binary search script invokes the Vivado backend iteratively and independently for each device to determine the minimum clock period constraint for which a legal solution can be found. For the Vivado flow where both the front- and the back-end can accept a specific period constraint, both stages are executed on each iteration by the binary search script. Lastly, to obtain the maximum clock frequency as well as the area utilization, a design analysis was performed by Vivado.

To allow for a more general conclusion, we target devices from two different FPGA generations: an Artix-7 (XC7A200) belonging to the 28nm 7-Series family and an Virtex UltraScale (XCVU440, 20nm) as our target devices. Note that Yosys uses a generic synthesis recipe for all AMD/Xilinx devices tuned primarily for (and in the case of Yosys-ABC9, with realistic delays exclusively for) the 7-Series family of devices [2] to which the Artix-7 device belongs. Given that our UltraScale target belongs to a different device family on a different technology node for which realistic primitive delays are not available, some loss in performance can be expected.

The benchmark designs stem from Verilog-to-Routing (VTR) [4], some of which overlap with those studied in [1]. We edited the designs to include previously omitted single and dual port RAM modules, and made other minor modifications to fix broken designs that could not be synthesized. The final designs with our modifications can be found in our Git repository. To improve comparability, we also tested the performance on additional designs used in [1], namely *des50* and *AES x3*. To cross-validate the performance against designs from a more diverse set of sources, we measured performance on designs from OpenCores [9].

Finally, to verify that Yosys-ABC9 indeed produces correct mappings, we perform a functional simulation of Yosys-ABC9 mappings for selected designs and check it against the behavior of the original HDL design file. We compared the output signals for *and_latch* and *stereovision2* for 100,000 randomly generated inputs. Stereovision2 was chosen because of its outsized improvement under ABC9 compared to Vivado.

V. Results and Discussion

We collected results for 3 sets of designs: the designs from "Mind The Gap" [1], the de-facto VTR benchmarks [4], and 18 designs from OpenCores [9] used in prior ABC studies.

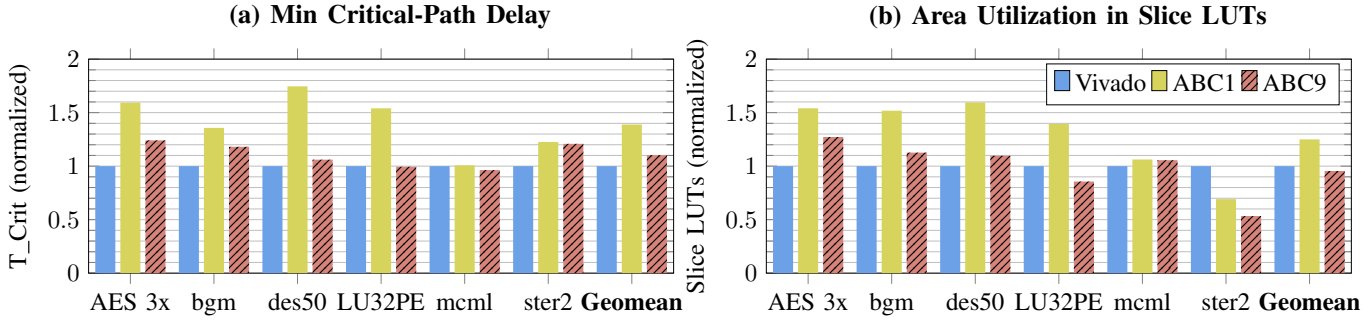


Fig. 6: Re-evaluation of the designs used in [1], which were the basis to demonstrate the 31% gap in 2015. As can be seen in (a), ABC9 has drastically improved the achievable critical path delay, leading to a new synthesis gap of 10% for this particular selection of designs.

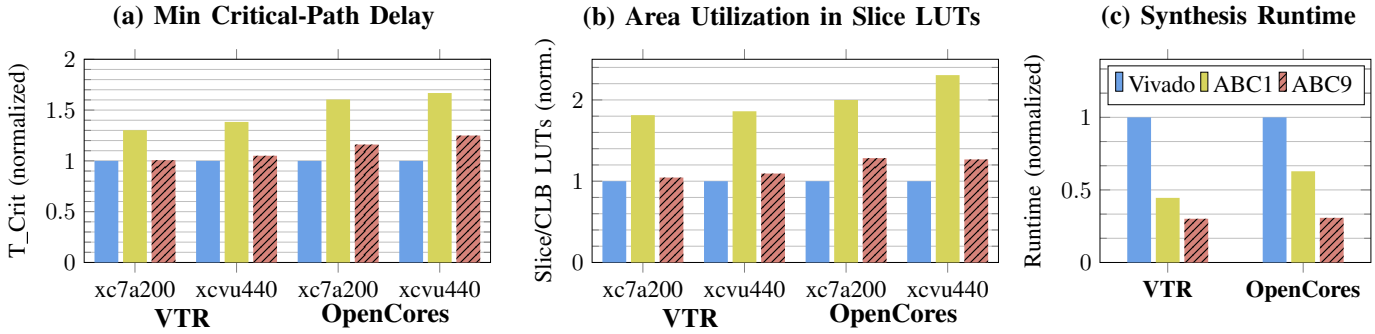


Fig. 7: Results Overview: **(a)** shows the normalized geometric mean of the minimum critical-path delay. **(b)** shows the normalized number of LUTs used corresponding to the mapping found in (a). **(c)** describes the runtime of the synthesis portion of each flow. (ABC1 and ABC9 do refer to Yosys-ABC1/9.)

Figure 6 shows minimum critical path delay and the corresponding area utilization in Slice LUTs for the designs from [1], on the 7-Series target. We can clearly see that Yosys-ABC9 is a substantial improvement over Yosys-ABC1 in both metrics, reducing the average delay gap from 31% to 10%. For area utilization, 2 out of 6 designs yield smaller mappings with ABC9, resulting in 4% less area utilization than Vivado on average.

Figure 7 summarizes the results of the VTR and OpenCores designs for both target FPGAs, and includes runtime measurements. Due to space constraints, we present the geometric mean of all benchmarks in each set, normalized against the Vivado baseline; exact values can be found on our GitHub repository [8]. The VTR benchmark results continue the trend we observed in Figure 6. For the 7-Series target, the Vivado-Yosys delay gap disappears, dropping from 30% (ABC1) to 0%. The improvement in area utilization is equally drastic, with the gap dropping from 81% to 4.4%. For the UltraScale target, the delay and area gap drop to 5% and 9% respectively. For the OpenCores designs the delay and area gap range around 15–25%. In terms of runtime, Yosys-ABC9 only uses a third of the runtime that Vivado spends on synthesis for both set of designs.

Evidently, ABC9 does not perform as well on the UltraScale target as it does on the 7-Series target. We believe the main factor for this inefficiency that Yosys-ABC9 employs realistic logic delays available only for the 28nm 7-Series family, and which do not exist for the 20nm UltraScale family. Effectively, our UltraScale experiment is asking Vivado to place-and-route a circuit synthesized against 28nm delays onto a 20nm device, on which both the absolute logic delay values, as well as the relative logic and routing delay ratios, will be different leading to a potential handicap.

Note that of the 26 VTR benchmarks, we chose to omit designs *and_latch* and *multiclock_separate_and_latch* because of their simplicity. (Both designs yielded the same results with all 3 flows). The design *boundtop* has a multiple driver bug and could not be synthesized. Design *mkDelayWorker32B* synthesized with good results for Yosys-ABC9, but had too many errors in the source RTL to be verified with the simulator, and was subsequently omitted. We present results using the remaining 22 benchmarks. For 3 of these designs (*diffeq1,2* and *mcml*) a minor bug meant it was necessary to disable aggressive DSP optimizations (i.e. register packing) for ABC9.

VI. Suggested Next Implementation Steps

The following implementation suggestions are in the approximate order of expected payoff.

A. Sequential Mapping

Traditionally, mapping and retiming are separate synthesis steps performed sequentially. It may be possible to implement a "sequentially transparent" mapping [10], which allows the register boundary to be determined dynamically to reflect the structure of the LUTs selected by the cut-based mapper [11, Section 5]. This sequential mapping can decrease path

delay by 20%, compared to performing mapping and retiming separately, while keeping the area penalty small (typically, less than 3 percent).

Furthermore, various other tradeoffs can be explored, for example, when the sequential mapping is only applied to the "sequentially critical" path. It should be noted that the current implementation of retiming in ABC1 and ABC9 is very limited, increasing the expected payoff of the sequential mapping.

B. Resubstitution Engine

More powerful resubstitution engines can be developed and used during both technology-independent and technology-dependent states of synthesis. The engine can support a wider set of Boolean transforms, solved by a variety of versatile transform solvers. This may translate into stronger engines for AIG rewriting, don't-care based optimization, and even equivalence checking, further enhancing the open-source synthesis flow.

C. Structural Choices with Boxes

Structural choices used in the current version of ABC9 are purely combinatorial in the sense that inputs to a choice node are two different logic node structures. In the future, the use of structural choices can be extended to work with box implementations, in addition to the combinatorial logic. For example, one may use a choice node to allow for implementing an adder using random logic (LUTs) or a predefined carry-chain (box), which is expected to further improve the quality of results produced by the mapper.

D. Synthesis Recipes

The order of boolean optimization passes, often referred to as synthesis *recipes*, can make a substantial difference to the post-synthesis quality of results. As a result, several recent works have explored black-box optimization [12], machine learning (ML) [13], and reinforcement learning (RL) based approaches [14], [15] to tune these recipes. Note that research on synthesis recipe tuning is only made possible by open-source tools that offer direct control and visibility over optimizations applied to the design. With improvements from ABC9, it is further likely that future work on optimal recipe search based on Yosys-ABC9 could produce synthesis results close to, if not out-performing, fixed Boolean optimizations by industrial tools.

E. Area-Delay Trade-off

By default, ABC1 and ABC9 are asked to find the minimum-depth or -delay mapping solution, with the expectation that doing so relieves timing-closure pressure on the placement and routing stages that follow. In cases where this pressure does not exist (for example, when not attempting to find the minimum possible clock period) it would be desirable to instead trade-off increased delay for decreased area. The ability to do this exists in the timing-aware ABC9 flow with support for an explicit delay target (as opposed to best-possible delay target). However, the challenge here would be to predict a realistic delay target that is neither too aggressive, leaving too much to be done at the back-end, nor too conservative and thus wasting area.

VII. Conclusion

With the improvements implemented in Yosys-ABC9, open-source FPGA logic synthesis and technology mapping is now competitive with state-of-the-art proprietary tools, reaching similar performance for both clock frequency and area utilization, while using a third of the runtime. Yosys-ABC9 has been successfully integrated into the (latest) upstream version of Yosys which is available under the open-source ISC license (equivalent to the simplified BSD license in being permissive thus allowing for commercial usage) and can be obtained from https://github.com/YosysHQ/yosys

Furthermore, the exact scripts necessary to reproduce the presented results are available at our GitHub repository [8] for others to validate and build upon. We hope that this tool can be the basis of further research into both FPGA and ASIC synthesis.

References

[1] E. Hung, "Mind The (Synthesis) Gap: Examining Where Academic FPGA Tools Lag Behind Industry," in *2015 25th Int. Conf. on Field Programmable Logic and Appl. (FPL)*, pp. 1–4, 2015.

[2] C. Wolf, "Yosys manual." http://yosyshq.net/yosys/documentation.html.

[3] R. Brayton and A. Mishchenko, "ABC: An Academic Industrial-Strength Verification Tool," in *Computer Aided Verification* (T. Touili, B. Cook, and P. Jackson, eds.), pp. 24–40, Springer, 2010.

[4] K. E. Murray, O. Petelin, S. Zhong, J. M. Wang, M. Eldafrawy, J.-P. Legault, E. Sha, A. G. Graham, J. Wu, M. J. P. Walker, H. Zeng, P. Patros, J. Luu, K. B. Kent, and V. Betz, "VTR 8: High-Performance CAD and Customizable FPGA Architecture Modelling," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 13, may 2020.

[5] H.-T. Zhang, J.-H. R. Jiang, L. Amarú, A. Mishchenko, and R. Brayton, "Deep Integration of Circuit Simulator and SAT Solver," in *2021 58th ACM/IEEE Design Automation Conf. (DAC)*, p. 877–882, IEEE Press.

[6] E. Lehman, Y. Watanabe, J. Grodstein, and H. Harkness, "Logic Decomposition during Technology Mapping," in *1995 IEEE/ACM Int. Conf. on Computer-Aided Design*, ICCAD '95, p. 264–271, IEEE.

[7] S. Chatterjee, A. Mishchenko, R. Brayton, X. Wang, and T. Kam, "Reducing Structural Bias in Technology Mapping," in *ICCAD-2005. IEEE/ACM Int. Conf. on Computer-Aided Design, 2005.*, pp. 519–526.

[8] "Our Git Repository for This Paper." https://github.com/growly/date23_narrowing_the_gap.

[9] OpenCores:. OpenCores.org.

[10] P. Pan and C.-C. Lin, "A New Retiming-Based Technology Mapping Algorithm for LUT-Based FPGAs," in *1998 ACM/SIGDA Sixth Int. Symposium on Field Programmable Gate Arrays*, (New York, NY, USA), p. 35–42, Association for Computing Machinery.

[11] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton, "Combinational and sequential mapping with priority cuts," in *2007 IEEE/ACM Int. Conf. on Computer-Aided Design*, ICCAD '07, p. 354–361, IEEE Press, 2007.

[12] A. Grosnit, C. Malherbe, R. Tutunov, X. Wan, J. Wang, and H. B. Ammar, "BOiLS: Bayesian Optimisation for Logic Synthesis," *arXiv preprint arXiv:2111.06178*, 2021.

[13] C. Yu, H. Xiao, and G. De Micheli, "Developing Synthesis Flows without Human Knowledge," in *55th Annual Design Automation Conf.*, pp. 1–6, 2018.

[14] X. Timoneda and L. Cavigelli, "Late Breaking Results: Reinforcement Learning for Scalable Logic Optimization with Graph Neural Networks," in *2021 58th ACM/IEEE Design Automation Conf. (DAC)*.

[15] A. Hosny, S. Hashemi, M. Shalan, and S. Reda, "Drills: Deep Reinforcement Learning for Logic Synthesis," in *2020 25th Asia and South Pacific Design Automation Conf. (ASP-DAC)*, pp. 581–586, IEEE, 2020.